Encapsulated Path Planning for Abstraction-Based Control of Multi-Vehicle Systems

Venkatesh G. Rao, Tichakorn Wongpiromsarn, Thientu Ho, Kimberly Chung and Raffaello D'Andrea

***Abstract*— We introduce an encapsulation approach to path planning for multi-vehicle control that permits other decision processes to work with simple, spatially-abstracted domain models, and partially or wholly ignore obstacles, observation uncertainty and vehicle dynamics. The encapsulation results in formal domain representations called *patch models* that generate path planning problems in two standard forms: *patch realization* and *differential patch realization*, that we formulate and address in this paper. We present enhancements to existing ways of partitioning the path planning problem, and also present enhancements for each component. Sufficient conditions are presented showing that domains with a scale-free relation between obstacle sizes and inter-obstacle separations are suitable for abstraction-based approaches. Application to multi-vehicle command and control is discussed.**

I. INTRODUCTION

Path planning is a kernel control and computation process in multi-vehicle systems. Other decision processes, such as coordination, scheduling and planning must define, constrain or make assumptions about vehicle motion to achieve their effects. We present an *encapsulation* approach that permits other decision processes to encode their path-planning needs in a simple, standard form, and partially or completely avoid reasoning about obstacles, dynamics or measurement uncertainty. This work is a component of ongoing research in distributed command and control [1] and has several sub-components, which are described in detail in previous [2] and forthcoming [3], [4] papers. The goal of this paper is to provide an integrated system-level description of the technical approach, summarizing component technology and treating system-level issues not addressed elsewhere.

Encapsulation is a design principle in system architecture [5] that achieves openness and robustness by hiding complex subsystems under an abstraction layer. Familiar examples are *packets* in communication and *objects* in programming. Complex systems, characteristically [6], display coupling between subsystems, making perfect encapsulation difficult. Significant benefits, however, are realizable through partial encapsulation.

Here, we seek to encapsulate a *controlled* function: vehicular motion. Our approach is based on spatial abstraction, and permits upstream decision processes, in particular human interaction and AI processes, to use a domain model that is much simpler than the one used for detailed path planning. The resulting formalism of *patch models* is discussed in a companion paper [1] and has been implemented in a prototype multi-vehicle command and control system, *Patchworks* (Fig. 1). In this paper, we address path-planning problems generated by such abstraction-based systems. The problems are also of interest in their own right, since they belong to the insufficiently-studied class of path planning problems with time-dependent state constraints. The approach is suitable for domains, such as deserts and sparse forests, that have certain *abstractability* properties that are characterized later in the paper.

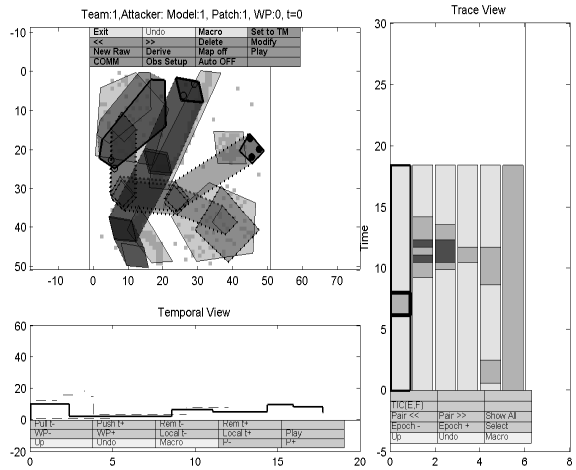

Fig. 1. Screenshot of *Patchworks* for a single distributed command and control (C2) node, showing spatially abstracted view (top left), patch velocity profiles (bottom) and patch interactions (left). Patches (moving polygons) represent both vehicle teams and terrain features.

Path planning has fascinated researchers since Euler solved the first non-trivial case, the *Bridges of Konigsberg* problem, in 1736, and has achieved canonical status in at least four fields. In control theory, it takes the form of the standard two-point boundary value problem (TPBVP) [7]. In robotics [8], the *Piano Mover's* formulation is central [9]. In operations research [10], shortest-path graph algorithms are fundamental, and in computer science, the classical problem is that of the *pebble automaton* [11]. Problem cases may vary from trivial to PSPACE hard [8], [12], [13].

The focus in this paper is on path planning *systems* rather than particular cases of the path planning *problem*. A path planning system must solve all the instances of problems that may occur in a particular application domain. The domains of interest in this paper are *multi-vehicle* domains, such as battlefields. These domains require *predictability* in motion to enable coordination among vehicles, and between vehicles

This work was supported by the AFOSR MURI on Cooperative Control

The authors are with the School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY 14853. Correspondence: vr47@cornell.edu

and humans. Good predictability is hard to achieve in the presence of unknown obstacles, poor communication and mutual observation constraints. In our work, we address these factors using *spatial abstraction*. Abstraction-based decision processes [14] lead to problems of *realization*. A realization process incorporates into partial action specifications the details that were ignored in abstract decision-making. Realization problems are the concern of this paper.

The original contribution of this work comprises three elements. First, we present the first systematic encapsulation of the path planning function suitable for multi-vehicle systems. Second, we make several enhancements to both the system and component-level design of decomposition-based path planning methods. Finally, we provide a precise characterization, through sufficient conditions, of domains that support abstraction-based path planning.

II. PATCH REALIZATION PROBLEMS

Patch models are computer representations of two-dimensional multi-vehicle domains that use a primitive object, the *patch*, to represent all entities. Patches are polygons that move with piece-wise constant velocities along piece-wise linear trajectories. Vehicles controlled by a particular command and control (C2) decision node must track the patch containing them, as they move (Fig. 1). Humans and AI processes achieve their objectives by manipulating *command reference models* built from patches. Informally, patch models permit vehicle control to be as simple as drag-and-drop operations with icons on a GUI such as Fig. 1. Under favorable conditions, patches can pass "through" obstacles and hide the effects of vehicle dynamics and uncertain observations. This capability yields simple yet expressive abstract models with sufficient predictive power to support a rich variety of client decision processes [1].

We consider two standard-form *patch realization problems*, instances of which are generated by an evolving abstract patch model (Fig. 1). The first problem is to compute a vehicle trajectory that stays within the *trace* of the moving polygon and satisfies end-time constraints. The second, harder problem, is to compute a trajectory such that the vehicle is inside the moving polygon at *every* instant. Formally, a patch $P(t)$ is a convex polygon translating with steady velocity, $V_{\rm nom}$ in the plane, defined by vertices $[x_1(t), y_1(t)), \ldots, (x_m(t), y_m(t)]$, over a time interval $[t_0, t_f]$ (Fig. 2). We refer to the instantaneous polygon as $P(t)$ and define the *trace* $P([t_1, t_2])$ as:

$$P([t_1, t_2]) = \bigcup_{t \in [t_0, t_f]} P(t) \tag{1}$$

Patch Realization (PR): *Let D be a connected subset of the plane with obstacles represented by closed, simply-connected sets $R_1, \ldots R_n$ in D. Let (x_i, y_i) and (x_f, y_f) be points in the free space, $\mathcal{F} \equiv D - (R_1 \bigcup \ldots \bigcup R_n)$. Let $P(t)$ be a patch such that $(x_i, y_i) \in P(t_0)$ and $(x_f, y_f) \in P(t_f)$. Find a trajectory*

$$\begin{array}{rcl} \phi : [t_0, t_f] & \rightarrow & \mathcal{F} \bigcap (P([t_0, t_f]) \\ \phi(t_0) & \in & P(t_0) \\ \phi(t_f) & \in & P(t_f) \end{array} \tag{2}$$

realizable by the center of mass of a vehicle governed by dynamic model Σ.

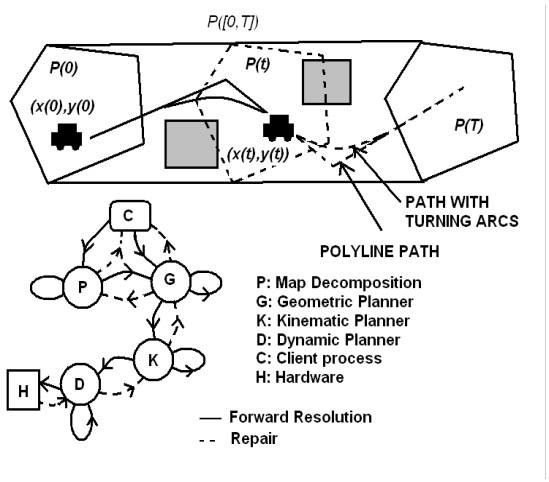

Fig. 2. Realization problems and solver architecture. Solving PR and DPR can be informally understood as satisfying 'group, drag and drop' vehicle motion commands issued by humans or automated processes, encoded by the moving polygon above.

PR requires the vehicle to stay inside the trace of the patch at all times, and additionally, inside the instantaneous position of the patch at start and end times. The complete problem formulation depends on vehicle model Σ. PR is a suitable formulation for problems where a transit corridor must be marked off in advance for the duration of the mission to prevent conflict with other vehicles (such as in friendly-fire prevention on battlefields). In PR, bounded spatial deviations are acceptable, and in-mission delays are acceptable so long as they are eventually compensated for. In crowded multi-vehicle domains with tighter deconfliction or coordinated motion constraints, even in-mission delays are not acceptable, and this gives us the *differential patch realization problem:*

Differential Patch Realization (DPR): *Given a patch realization problem, find a solution ϕ that satisfies time-varying constraint*

$$\phi(t) \in P(t) \text{ for all } t, \tag{3}$$

in place of the constraints in Eqn. (2).

Solutions to a DPR problem solve the corresponding PR problem, but not vice-versa. Both are illustrated in Fig. 2. The formulation does not commit to specific vehicle characteristics or specific levels of pre-mission and in-mission information availability. Problem instances can therefore vary in complexity according to these factors. The methods described in this paper are suitable for PR problems involving car-like robots, and DPR problems for cases where dynamics

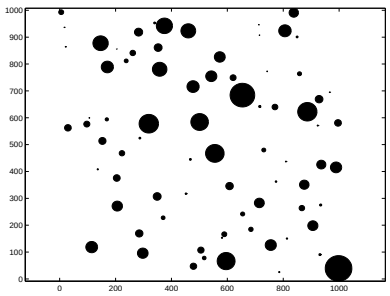

Fig. 3. Abstractable domain constructed according to sufficient condition in proposition 1 (appendix)

No.	State Trace	Instance type
1.	CGDH	One-pass PR solution
2.	CGKDH	One-pass DPR solution
3.	CGPGDH	PR with in-line map refinement
4.	CGDGDH	PR with geometric repair
5.	CGDHDGDH	Execution failure with geometric repair
6.	CGDGCGDH	Failure with repeat client invocation
7.	CGDDH	Local dynamic repair
8.	CGPCPGD	Human-guided map processing

TABLE I

TYPICAL PROCESS TRACES IN PATH-PLANNING SUBSYSTEM

are trivial. Both full and partial pre-mission information cases can be handled. PR and DPR cannot be solved in a vehicle or domain independent way[1], but they can be *specified* without regard to either, providing the basis for encapsulation.

For clarity, discussion of existence conditions for PR and DPR is in the appendix, where we present a sufficient condition for simple vehicles, showing that solutions for PR exist in domains where obstacle sizes are related to inter-obstacle distances in a scale-independent way. An example domain constructed using this condition is in Fig. 3.

III. DECOMPOSITION-BASED SOLUTION

As stated, PR and DPR are path planning problems with time-dependent state constraints. A common partitioning of the path planning problem [8] comprises map preprocessing, geometric planning and kinodynamic planning. In this paper, we extend this partitioning using components for *in-process map decomposition*, *geometric planning*, *kinematic planning* and *dynamic planning*, that are discussed separately in the subsections that follow. The separation of kinematics and dynamics and the allowance for in-process map decomposition provide the enhanced control authority required in multi-agent applications. The solution process can be described by the state diagram in Fig. 2, where nodes represent component solvers. A taxonomy of process types can be constructed using this state diagram, and typical cases are listed in Table I. To solve PR, kinematics need be considered only to the extent required in order to satisfy boundary constraints, and the 'K' state can be omitted. In the next 4 subsections, we discuss component-level innovations corresponding to the computations required at nodes P, G, D and K. System-level behavior is discussed in subsection III-E.

A. Adaptive, Human-Guided Map Decomposition

Map decomposition (more generally, configuration space decomposition) is the process of generating a graph that conforms to the free-space topology of a domain with obstacles. A path found with shortest-path graph search is only as good as the underlying decomposition. Voronoi diagrams and cellular decomposition [8] are the two popular

[1]it is trivial to construct terrains (such as spiral mazes) or vehicle models (such as a high turn-radius aircraft) to render any particular patch realization problem infeasible. The intent in this paper is to take advantage of the large number of cases where the realization problems *are* solvable.

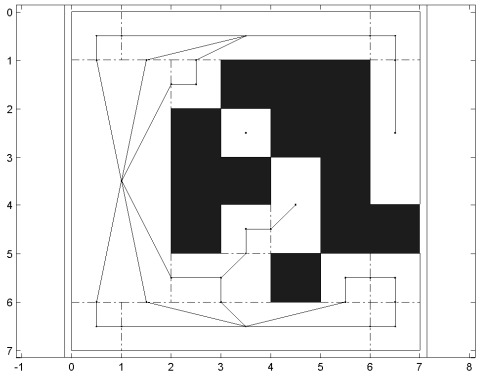

Fig. 4. Coarse, minimal decomposition and small connectivity graph

decomposition methods. In the latter, which we adopt, the free space is decomposed into a number of cells with a simple geometry. Our method, which uses variable-sized rectangles, has three special features. First, it is constructed to be used *during* path planning (case 3 in Table I), as well as during pre-processing. The initial, minimal decomposition can be locally improved in real-time to suit the problem. Second, the method allows humans (or AI processes) to guide the decomposition process (case 8 in Table I), by forcing it to conform to artificial constraint boundaries. Third, the method supports partitioning of very large maps into *sectors*, by treating the boundary of each sector in a special way to allow for 'pasting' of small maps into larger maps. This provides support for distributed processing and localized map management.

The detailed description of the decomposition algorithm is in the appendix. Here, we briefly illustrate the first two features. In-process resolution improvement is illustrated in Figs. 4 and 5. For unconstrained problems, the minimal decomposition is sufficient. For our application, the decomposition must be locally fine enough to solve the PR problem.

The effect of human guidance in map decomposition is shown in Figs. 6 and 7. In this case, with just 5 human-added lines, the cluster of obstacles is efficiently isolated, which leads to higher-quality paths on the resulting graph. Other uses of this facility include creating decompositions to conform to patch boundaries or virtual fences.

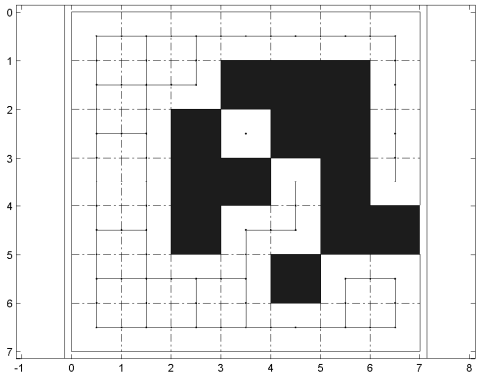

Fig. 5. High resolution decomposition and large connectivity graph

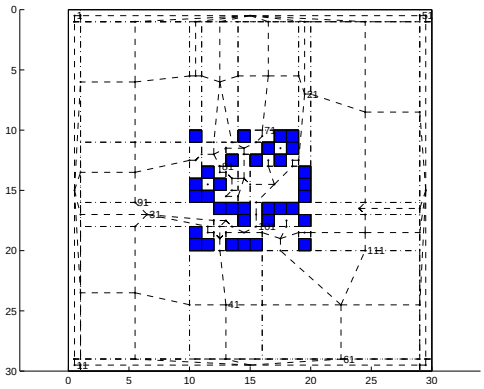

Fig. 6. Fully automated decomposition, note poor cluster isolation

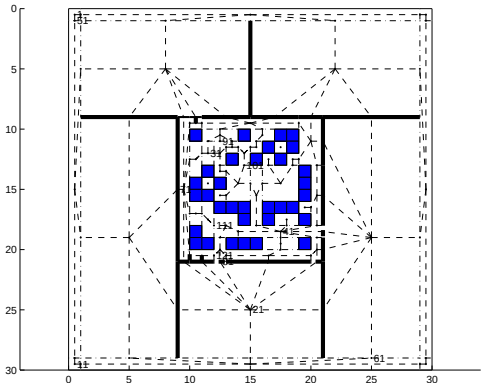

Fig. 7. Human-guided decomposition: bold lines placed by operator to isolate cluster

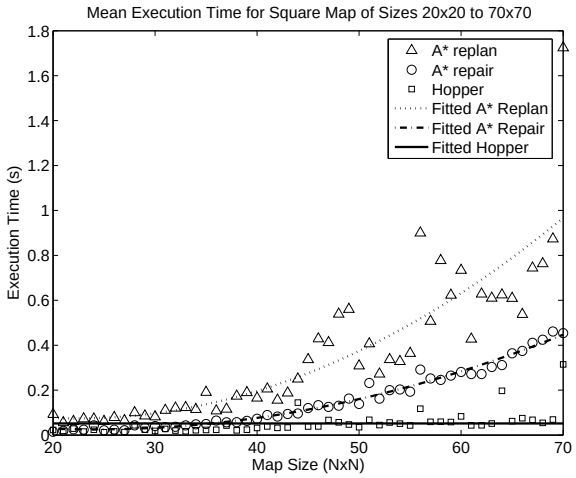

Fig. 8. Performance of *Hopper* algorithm for local geometric repair

B. Geometric Refinement and Repair

Once a suitable decomposition has been created, a path, ϕ, must be found and passed on to the kinematic and dynamic planners, which may return the solution for repair if it is found to be infeasible. The initial path planning problem is well-studied, and little innovation is required there. Our implementation uses the shortest-path algorithm, A^* search, on the portion of the connectivity graph that is within the trace of the patch. The only modification required is the implementation of a method to prune the graph spatially (we use Matlab's *inpolygon* function).

The non-trivial problem is that of repair, which happens when an edge of the graph on an initial solution ϕ is rendered infeasible. A repair problem instance is generated when a downstream resolution algorithm fails or an execution failure occurs (for example, process cases 4 and 5 in Table I). We developed a lightweight, complete repair method called the *minimum-deviation hopper* that solves the repair problem by deleting the failed edge from the graph and searching locally for a way around it. *Hopper* performs remarkably well compared to both replanning from scratch and A^* repair. Simulations were conducted with a range of map sizes from 20×20 to 70×70, with 30 problem instances in each case. The results, in Fig. 8, show that computation time tends to stay flat with increasing map size for *Hopper*, while increasing for the other two methods. This makes the method suitable for real-time implementation on small robots with low processing/memory capabilities. The main advantage, compared to existing repair approaches such as D^* [15] is that *Hopper* attempts to stay near the original path, making it suitable for PR and DPR.

Hopper was also evaluated in domains with moving obstacles, where the repair problems are generated due to execution failure (case 5 in Table I) arising from temporary blockage of a graph edge. The moving obstacles were assumed to be neutral, randomly-moving entities that were assumed to have their own obstacle avoidance mechanisms. *Hopper* was compared to a simple wait-till-clear strategy for various ratios of vehicle to obstacle speed. As might be expected, it makes sense to repair a path if the moving obstacles are much slower than the vehicle, and to wait otherwise. This behavior is illustrated in Fig. 9, which shows mission time vs. relative obstacle speeds for both *Hopper* and wait-till-clear strategies. Further details of this work are in a forthcoming paper [4].

C. Kinematics: The Spotlight Tracking Problem

Given a path, ϕ, on a graph that is within the patch trace, a velocity planner must assign velocities to each leg

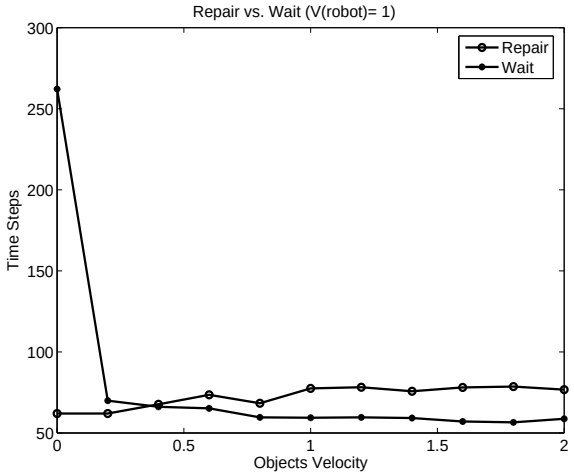

Fig. 9. Mission time comparison: *Hopper* repair vs. wait-till-clear strategies among moving obstacles

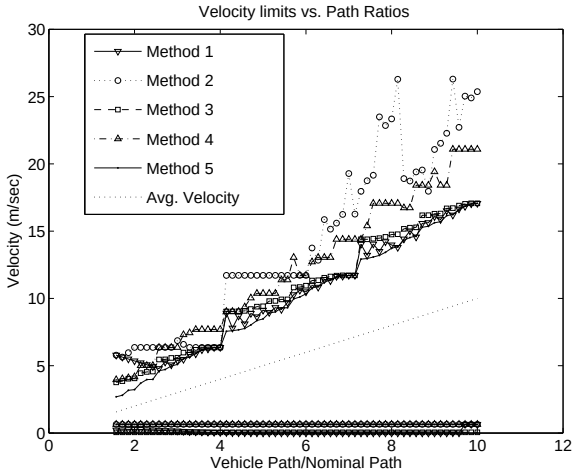

Fig. 10. Velocity limits for profiles generated using five different methods; $V_{\rm nom} = 1$

of ϕ. For DPR the velocity profile must, in general, be time-varying. This problem has, to our knowledge, not been studied. It falls into the class of path planning problems with continuously time-varying state constraints. We refer to this problem as the *Spotlight Tracking Problem* (STP), since it can be visualized in terms of an agent attempting to stay inside a moving spotlight while avoiding obstacles. STP can be stated as follows:

Spotlight Tracking Problem: *Given a poly-line path ϕ of length $|\phi|$, within patch trace $P([t_0, t_f])$, find $v([t_k, t_{k+1}])$, the magnitude of the vehicle velocity for each time step k such that the vehicle position $(x(t_k), y(t_k))$ is inside $P(t_k)$ for all k, where the patch $P(t)$ is moving at velocity $V_{\rm nom}$.*

The continuously-varying state constraint with a non-smooth boundary makes closed-form analysis of this problem extremely hard. The shortcomings of two obvious but naive approaches highlight the sources of complexity. Consider the following velocity-setting methods:

$$v([t_k, t_{k+1}]) = V_{\rm nom} \arccos(\theta - \theta_{\rm nom}) \quad (4)$$

$$v([t_k, t_{k+1}]) = V_{\rm avg} = |\phi|/(V_{\rm nom}(t_f - t_i)) \quad (5)$$

The method in Eqn. (4) attempts to speed up to compensate for the difference in polygon heading (direction of $V_{\rm nom}$) and vehicle heading (the direction of the current edge on ϕ). This method generates infeasible velocities for heading errors greater than $\pi/2$. The method in Eqn. (5) uses a constant average velocity based on the difference in nominal (patch centroid) and actual (ϕ, vehicle) path lengths. This satisfies PR, but fails for DPR due to non-uniform changes in path geometry (such as can occur at terrain transition points).

Based on preliminary investigations, two variables, L_1 and L_2, the *forward* and *backward* slack, representing the arc-lengths from vehicle position to the leading and trailing edges of the polygon, along ϕ, were found to provide the most natural means for control. Five heuristic methods were constructed using these variables, of which one performed significantly better than the others. This method (Method 5 in Fig. 10) has the governing equation:

$$\begin{aligned} v([t_k, t_{k+1}]) &= \frac{T_1 T_2}{\Delta t} \\ T_1 &= \left(\frac{2L_1(k+1)}{L_1(k+1) + v([t_{k-1}, t_k])\Delta t} \right) \\ T_2 &= \left(\frac{v([t_{k-1}, t_k])\Delta t + V_{\rm avg}}{2} \right), \end{aligned} \quad (6)$$

where $V_{\rm avg}$ is given by Eqn. 5, and Δt is the constant time step (an estimate of $V_{\rm avg}$ can be used if the length of path ϕ is not known in advance). At each time step, the equation sets the velocity for the next time step using a product of two terms, the first of which builds in sensitivity to local path geometry by using the forward slack and the distance traveled in the previous time step, while the second term builds in sensitivity to global path characteristics. The method can be shown to be uniformly bounded. Details of this method, as well as the other four methods (which use different subsets of available information) are in a companion paper [3] and are not discussed here. Figure 10 shows the maximum and minimum velocities produced by the five methods over a large number of increasingly tortuous paths. As can be seen, Eqn. 6 produces profiles with nearly linear increase in maximum path velocity with increase in the ratio of actual to nominal path lengths. Fig. 10 can be used as an inverse look-up table that sets patch velocity, $V_{\rm nom}$, based on vehicle limits and terrain complexity. A straightforward *repair* mode solution to STP, to adapt a failed solution (occuring due to failure in dynamic refinement for example) is to locally decrease the speed as far as the available backward slack allows. This problem is yet to be investigated.

D. Dynamic Refinement

Accommodating vehicle dynamics by constructing paths out of arcs and straight line segments is not a new idea [16], [17], [18], and using such paths to refine poly-line paths found on graphs has also been considered [19]. Existing

literature, however, focuses on minimum length or minimum deviation formulations. To solve patch realization problems, we require control authority over speed and mission time and applicability to real-time settings with limited look-ahead. To achieve this, we constructed a sequential arc-fitting procedure using a discounted look-ahead mechanism. The method produces fast and aggressive turns (with large radii) when the immediate horizon is benign, and becomes more cautious (smaller turning radii) when sharp turns are right ahead. The governing equations are:

$$\begin{aligned} \rho_i &= \rho_{\min} + k_i(\rho^i_{\max}, \rho_{\min}) \\ k_i &= w_\theta \sum_{j=i}^{i+n_i-1} c_0^\theta \lambda_\theta^{j-i} \theta_j + \qquad (7) \\ & \quad (1-w_\theta) \sum_{j=i}^{i+n_i-1} c_0^L \lambda_L^{j-i} L_j, \qquad (8) \end{aligned}$$

where ρ_i is the radius of the turning arc at the i^{th} vertex, and $\rho^i_{\max}, \rho_{\min}$ are radius limits (the upper limit being vertex-dependent), computed to incorporate clearance, centripetal acceleration and turn radius limits. The coefficient k_i determines the aggressiveness of the turn, and is computed as a weighted sum of the line segment lengths and turn angles for the next n_i turns, with discounting parameters λ_θ and λ_L. The parameter w_θ provides control over the weighting of future turn angles relative to segment lengths. L and θ are the normalized length and turn variables, and the c_0 coefficients normalize the sums. Details are in [2]. The main capabilities of the method, can be succinctly captured in two graphs. Figure 11 shows the tradeoff between failure risk and speed. Failure occurs most commonly due to an over-aggressive turn leading to violation of acceleration or turning constraints in the next step. A *local repair* method (the self-loop in the D node in Fig. 2) was implemented that operated by reducing the turning arc radii at and near failure locations. Repair reduces the realization percentage by 20-40% (Fig. 12). Details are in [2].

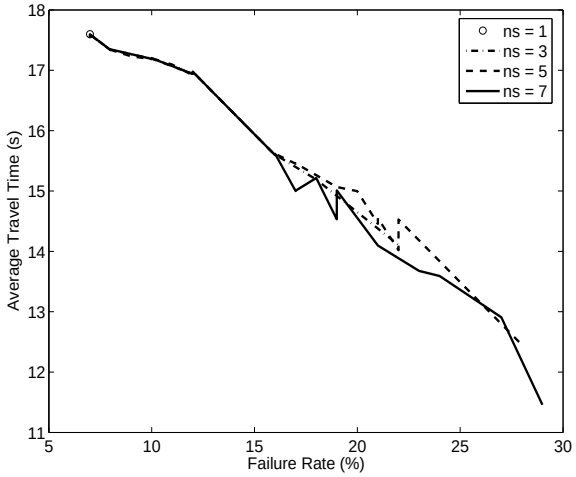

Fig. 11. Risk vs. failure tradeoff in aggressive maneuvering dynamic refinement

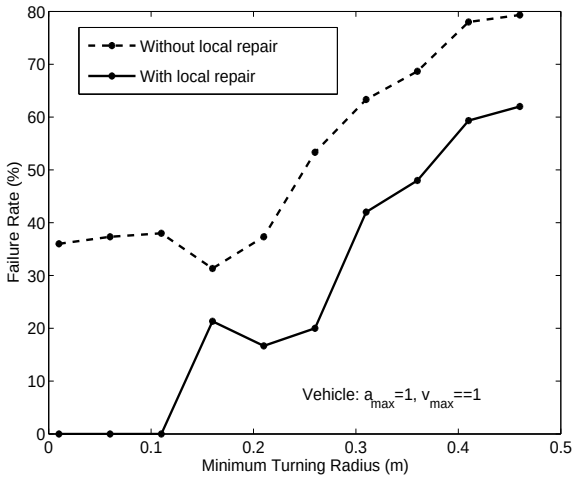

Fig. 12. Resolution success change with implementation of local repair

E. Subsystem Design

The problem of path-planning subsystem design has largely been studied with respect to specific platforms. Encapsulation provides one way to pose the problem formally with generality, in terms of a state diagram (Fig. 2):

Path Planning Subsystem Design: *Given the component interconnection state diagram, construct a state transition function* Π *such that for a class of domains* $\mathcal{D}$*, a class of vehicles* $\mathcal{V}$*, and a class of patch realization problems* $\mathcal{P}$*, every problem instance will be solved within* k *state transitions, with at most* r *transitions from the node* C*.*

The specification of Π (a repair mode selection rule) is non-trivial, and a host of related problems can be posed, concerning process convergence, live locks and dead locks, which require discrete-event systems tools [20]. While consideration of these issues is beyond the scope of this paper, we briefly illustrate them by considering the simplest possible process trace: processes of the general form $C(GD)^iH$ in

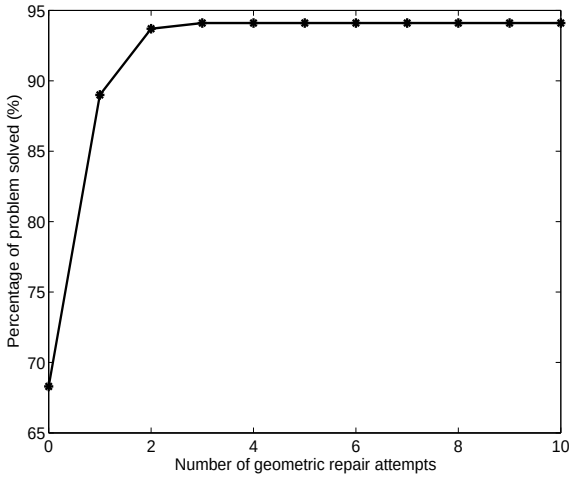

Fig. 13. Increasing resolution percentage in $C(GD)^kH$ iteration

the notation of Table I and Fig. 2. Figure 13 shows the results for a set of problem instances where the dynamic refinement fails initially for about 32% of the cases. Using the repair mode of the geometric module, the resolution percentage increases to about 93% at $i = 3$. In the remaining 7% of cases, more complex solution process traces are required. Here, ignoring the unresolved instances, for the worst case, $k = 2 \times i + 3 = 9$, of which 1 is a human input ($r = 1$).

To solve DPR the K component is required, and the D component must be capable of following time-varying velocity profiles. Currently, our dynamic planner can only vary *average* mission speed, and therefore DPR cannot be supported in general. The exception is the case of trivial vehicle dynamics. The subsystem design problem is an open-ended problem, the complexity of which depends on the size of the classes $\mathcal{V}, \mathcal{D}$ and $\mathcal{P}$ and the severity of the performance requirements r and k. The ratio r/k achieved is a measure of the amount of human oversight required for a given problem solving load, and is an important limiting factor for semi-autonomous multi-vehicle system design where the goal is to maximize human productivity.

IV. Conclusions

We presented an encapsulation approach for path planning in multi-vehicle domains and formulated the resultant standard-form problems. We presented a new partitioning of the problem and described component-level solutions for the map-partitioning, dynamic, and geometric components. We also introduced a new kinematic planning problem, the *spotlight tracking problem*, and solved it heuristically. Favorable domains were characterized with sufficient conditions showing that scale-free relations between obstacles and obstacle separations support abstraction. System integration results were presented for the simplest class of realization processes, and the general features of the system design problem were discussed.

Future work will focus on a more thorough examination of different solution process patterns, formal analysis of the system design problem, and integration with a real-time sensor-based map-building module.

References

[1] V. G. Rao and R. D'Andrea, "Patch models and their applications," in *American Control Conference (submitted)*, 2006.

[2] T. Wongpiromsarn, V. G. Rao, and R. D'Andrea, "Two approaches for dynamic refinement in hierarchical motion planning," in *AIAA Guidance, Navigation and Control Conference*, San Jose, CA, August 2005.

[3] K. Chung, V. G. Rao, and R. D'Andrea, "Predictable motion in unpredictable domains: The spotlight tracking problem," in *American Control Conference (submitted)*, 2006.

[4] T. Ho, V. G. Rao, and R. D'Andrea, "Geometric path planning," in *preprint*, 2005.

[5] D. Parnas, "On the criteria to be used in decomposing systems into modules," *Communications of the ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.

[6] H. Simon, *The Sciences of the Artificial (Third Edition)*. Cambridge, MA: MIT Press, 1996.

[7] A. E. Bryson and Y. C. Ho, *Applied Optimal Control*. Hemisphere-Wiley, 1975.

[8] J.-C. Latombe, *Robot Motion Planning*. Kluwer, 1991.

[9] J. T. Schwartz and M. Sharir, "On the piano mover's problem: I. the case of a two-dimensional rigid polygonal body moving amidst polygonal barriers," *Communications on Pure and Applied Mathematics*, vol. 36, pp. 349–398, 1983.

[10] C. Papadimitrou, *Combinatorial Optimization: Algorithms and Complexity*. Dover, 1998.

[11] M. Blum and D. Kozen, "On the power of the compass: Or, why mazes are easier to search than graphs," in *Proc. Annual Symposium on Foundations of Computer Science*, 1978, pp. 132–142.

[12] J. F. Canny, "The complexity of robot motion planning," Ph.D. dissertation, MIT, 1987.

[13] J. H. Reif, "Complexity of the mover's problem and generalizations," in *Proc. of the 20th IEEE Symposium on Foundations of Computer Science*, 1979, pp. 421–427.

[14] C. Knoblock, "Learning abstraction hierarchies for problem solving," in *Proc. 8th Nat. Conf. AI.*, Aug. 1990, pp. 923–928.

[15] A. Stentz, "Optimal and efficient planning for partially known environments," in *Proc. ICRA*, 1994.

[16] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, no. 3, pp. 497–517, 1957.

[17] J. A. Reeds and L. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific Journal of Mathematics*, vol. 145, no. 2, pp. 367–393, 1990.

[18] J. P. Laumond, P. E. Jacobs, M. Taix, and R. M. Murray, "A motion planner for nonholonomic mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 5, pp. 577–593, 1994.

[19] P. R. Chandler, S. Rasmussen, and M. Pachter, "UAV cooperative path planning," in *Proc. AIAA Guidance, Navigation and Control Conference*, Reston, VA, August 2000.

[20] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Kluwer, 2001.

Appendix

A. Realization Conditions

Let the two-dimensional domain D contain convex obstacles represented by closed, simply-connected regions $R_1, R_2, \ldots R_n$ in D. Let the rigid vehicle V have circumcircle diameter d_r and velocity-driven dynamics $\dot{x} = v_x$, $\dot{y} = v_y$. Let δR_i, the boundary of R_i, be piece-wise differentiable, with at most a finite number of corners. Let $|\cdot|$ represent Euclidean distance. Define obstacle diameter and separation as follows:

$$\begin{aligned} d_{\max}(R_i) &= \max_{p,q \in \delta R_i} |p - q|. \\ d_{\min}(R_i, R_j) &= \min_{p \in \delta R_i, q \in \delta R_j} |p - q| \end{aligned} \tag{9}$$

Let $R_1, R_2, \ldots R_n$ be ordered such that

$$d_{\max}(R_1) \geq d_{\max}(R_2) \geq \ldots \geq d_{\max}(R_n). \tag{10}$$

This ordering can always be achieved, since there are a finite number of obstacles. With these definitions, we state a sufficient condition for realizability (the complete proof is not presented due to space constraints).

Proposition 1 (Direct Realizability): *If, for all i, j, there exists a constant $\lambda > 0$ such that:*

$$\lambda d_{\max}(R_n) > d_r \tag{11}$$

$$d_{\min}(R_i, R_j) > 2\lambda \min(d_{\max}(R_i), d_{\max}(R_j)), \tag{12}$$

then all patch realization problems with the patch having instantaneous in-circle diameter greater than $\sqrt{2} d_{\max}(R_1)(1 + 2\lambda)$ are realizable.

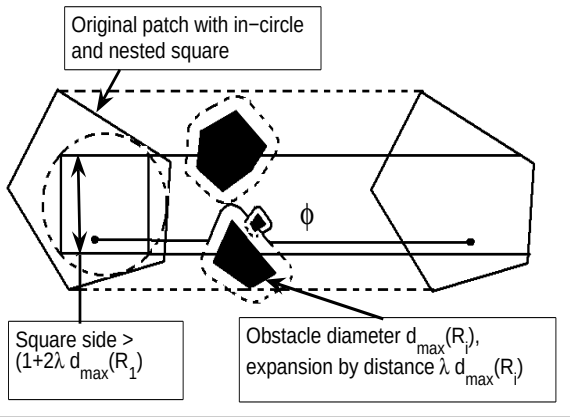

Fig. 14. Illustrative diagram for Proposition 1

Proof Sketch: We prove the result by induction. By the in-circle condition a square of side $d_{\max}(R_1)(1+2\lambda)$ and arbitrary orientation can be inscribed within any patch $P'([t_0,t_f])$. Let this derived patch, $P([t_0,t_f])$ be a translating square, oriented such that one pair of sides is parallel to the direction of translation. Let (x_s,y_s) and (x_f,y_f) be arbitrary start and end locations within the free regions of $P(t_0)$ and $P(t_f)$ respectively (the extension to P' is trivial).

For the induction hypothesis, assume that a piece-wise smooth curve ϕ with a finite number of corners exists, connecting (x_s,y_s) and (x_f,y_f), that is entirely within $P([t_0,t_f])$, and that does not intersect any obstacle with $d_{\max}(R_i) > d_{\max}(R_m)$, for some m. For each R_j, define R'_j to be the region generated by an outward normal vector moving along δR_j, with length $\lambda d_{\max}(R_j)$. R'_j has at most a finite number of discontinuities, by convexity of R_j. Let these be bridged by arcs of radius $\lambda d_{\max}(R_j)$ centered about the corresponding corner on δR_j. The region R'_j will have a diameter, defined as in Eqn. 9, of at most $(1+2\lambda)d_{\max}(R_j)$. Let there be $j=1,\ldots,k$ obstacles with diameter $d_{\max}(R_m)$, such that ϕ intersects $\delta R'_j$.

By the finite number of corners of ϕ, there can only be a finite number of intersections with $\delta R'_j$. Let $s_1(j)$ and $s_2(j)$ be the first and last points of intersection with R'_j, in the positive direction of traversal of ϕ. At least one of the two arcs $(s_1(j),s_2(j))$ on $\delta R'_j$ must be entirely within $P([t_0,t_f])$, by Eqn. 12. Replace $(s_1(j),s_2(j))$ on ϕ with this arc on $\delta R'_j$. Repeat for all k obstacles, and call the resulting curve ϕ'. By the separation condition (Eqn. 12), ϕ' cannot intersect any obstacle of diameter greater than $d_{\max}(R_m)$. Since k is finite, at most a finite number of new corners can be introduced into the curve.

Given the induction hypothesis above, we can construct a trajectory that avoids all obstacles by finding an initial one. This is trivial, since the largest obstacle has diameter $d_{\max}(R_1)$. The straight line, L connecting (x_s,y_s) and (x_f,y_f) has no corners and does not intersect any obstacle of diameter $d_{\max}(R_i) > d_{\max}(R_1)$, since none exist. By induction, a trajectory exists. By the construction and Eqn. 11, this path has minimum clearance $d_r/2$ at all points. Since the dynamics are single-integrator, this trajectory is realizable. QED.

Proposition 1 provides sufficient conditions for path existence in free space. We omit a similar *indirect realizability* property for graphs, due to space constraints. Domains constructed according to Proposition 1 have a scale-free appearance (Fig. 3), a result of the condition in Eqn. 12.

```
function [Rects, V, Adj]=Decompose(A,Rects,V,Adj, δ)

if Rects==EMPTY
% Initial decomposition
 [ObsCorners,ObsEdges]=FindEdgesCorners(A);
 BorderEdges=BorderProc(A,ObsEdges,ObsCorners);
 PartEdges=GetGuidance();
 Edges={ObsEdges,BorderEdges,PartEdges};
 for k=1: NumObsCorners
  if cornertype(ObsCorners(i))==OPEN;
   EdgeCornerChanges=RayProp(ObsCorners(i));
   Update(Edges, Corners,EdgeCornerChanges)
  endif
 endfor
 Rects=MakeRects(Edges,Corners);
 endif
Rects=Refine(Rects,δ);
(V,Adj)=GetGraph(V,Adj,Rects,Edges);
```

Fig. 15. Outline pseudocode for Adaptive Decomposition Algorithm showing main logical and procedural elements

B. Decomposition Algorithm

The adaptive decomposition algorithm is rather long, but much of the code is conceptually straightforward. We briefly describe the novel elements with reference to Fig. 15. Given a matrix A of zeros and ones, ones representing obstacles, the algorithm first detects all obstacle corners and edges (**FindEdgesCorners**). It then partitions a 1-unit wide border (**BorderProc**) of the map in a consistent way, to enable sector-wise handling of large maps by pasting along the border. **GetGuidance** then allows the operator to intervene and define guiding partitioning lines. Then, the automated decomposition takes over by visiting each convex corner in a loop and propagating a ray (**RayProp**) from it until it hits a pre-existing obstacle or partitioning line. It then adds and deletes appropriate edges and corners (**Update**). The process repeats until all open corners are taken care of. **MakeRects** and **RefineRects** create and refine rectangles by grouping corners and breaking up existing rectangles respectively, and **GetGraph** creates a graph by placing vertices at rectangle centers and open boundaries. It is straightforward to show that the decomposition algorithm Fig. 15 is complete. The proof relies on the fact that the number of convex obstacle corners in a map (non-convex free space corners) is finite, and monotonically decreasing with the number of ray-propagation steps in the main loop. Further details will be described in a forthcoming journal paper.